

EECS 370

Making Virtual Memory Fast



EECS 370

Making Virtual Memory Fast



Agenda

- **Table Look-aside Buffers (TLB)**
- Virtual Memory Walkthrough
- Cache Placement

Virtual Memory Performance

- To translate a virtual address into a physical address, we must first access the page table in physical memory
 - If it's an N-level page table, we must do N total loads before getting the physical page number
- Then we access physical memory again to get the data
 - A load instruction performs at least 2 memory reads
 - A store instruction performs at least 1 read and then a write
- Above lookups are **SLOW**

Translation look-aside buffer (TLB)

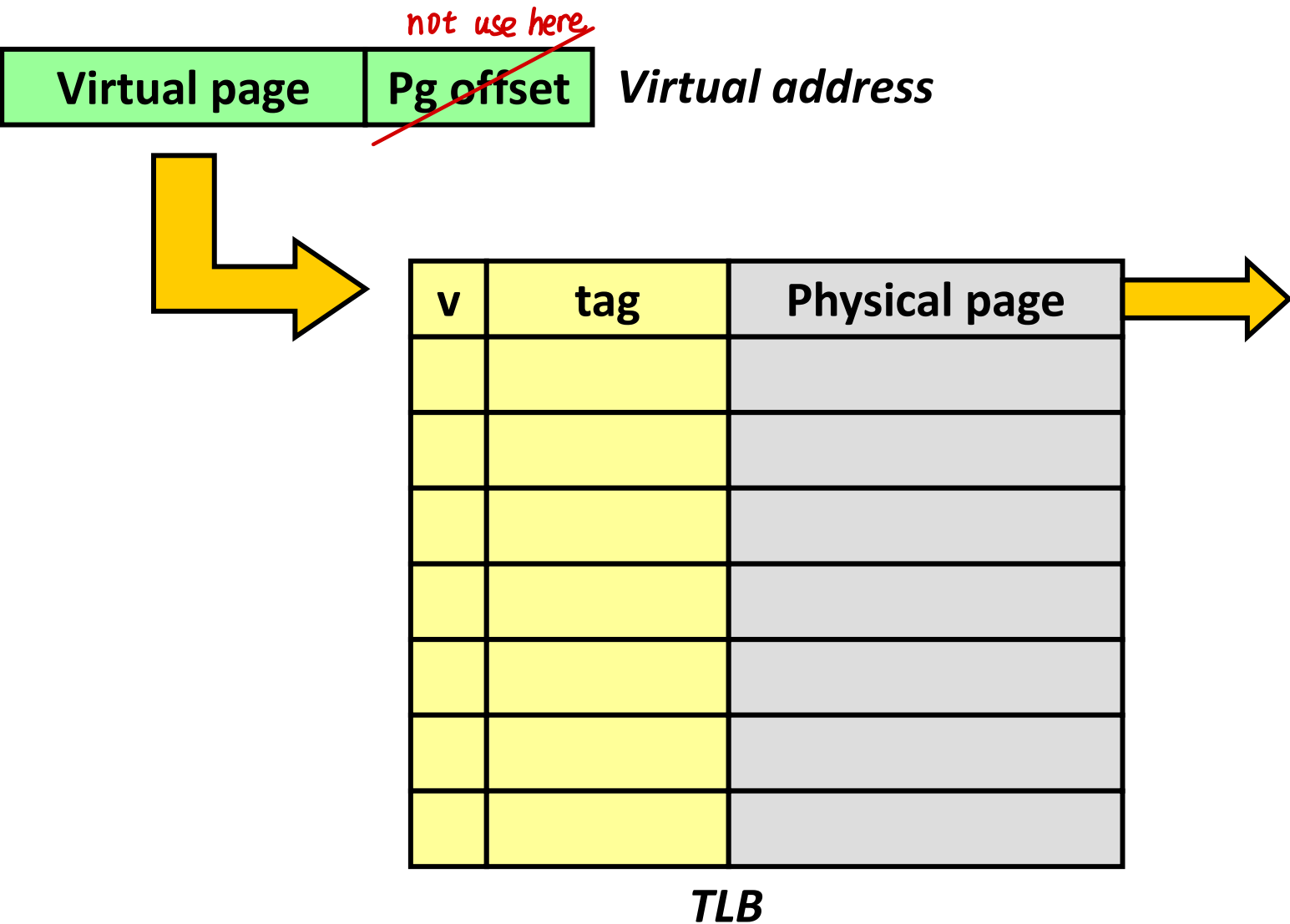
"specialized cache just to hold entries in the page table"

- We fix this performance problem by avoiding main memory in the translation from virtual to physical pages.
- Buffer common translations in a **Translation Look-aside Buffer (TLB)**, a fast cache memory dedicated to storing a small subset of valid V-to-P translations.
- 16-512 entries common.
- Generally has low miss rate ($< 1\%$).

Poll: Why can we make TLBs smaller than the cache hierarchy?

- a) Addresses are smaller than data
- b) TLB is accessed less frequently than caches
- c) ☒ Only need to store info about individual pages

Translation look-aside buffer (TLB)



Putting it all together

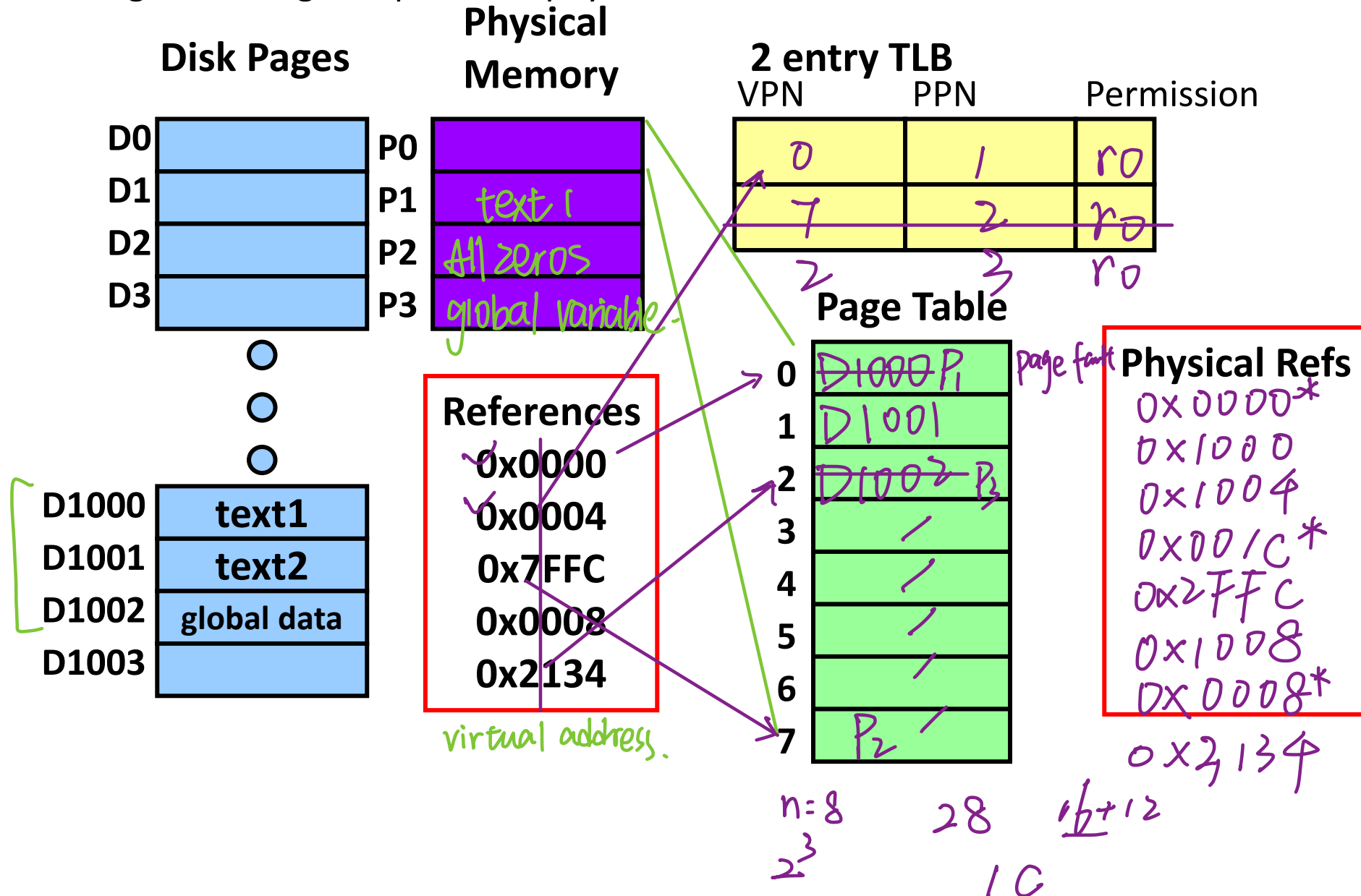
- Loading your program in memory
 - Ask operating system to create a new process
 - Construct a page table for this process
 - Mark all page table entries as invalid with a pointer to the disk image of the program
 - That is, point to the executable file containing the binary.
 - Run the program and get an immediate page fault on the first instruction.

Agenda

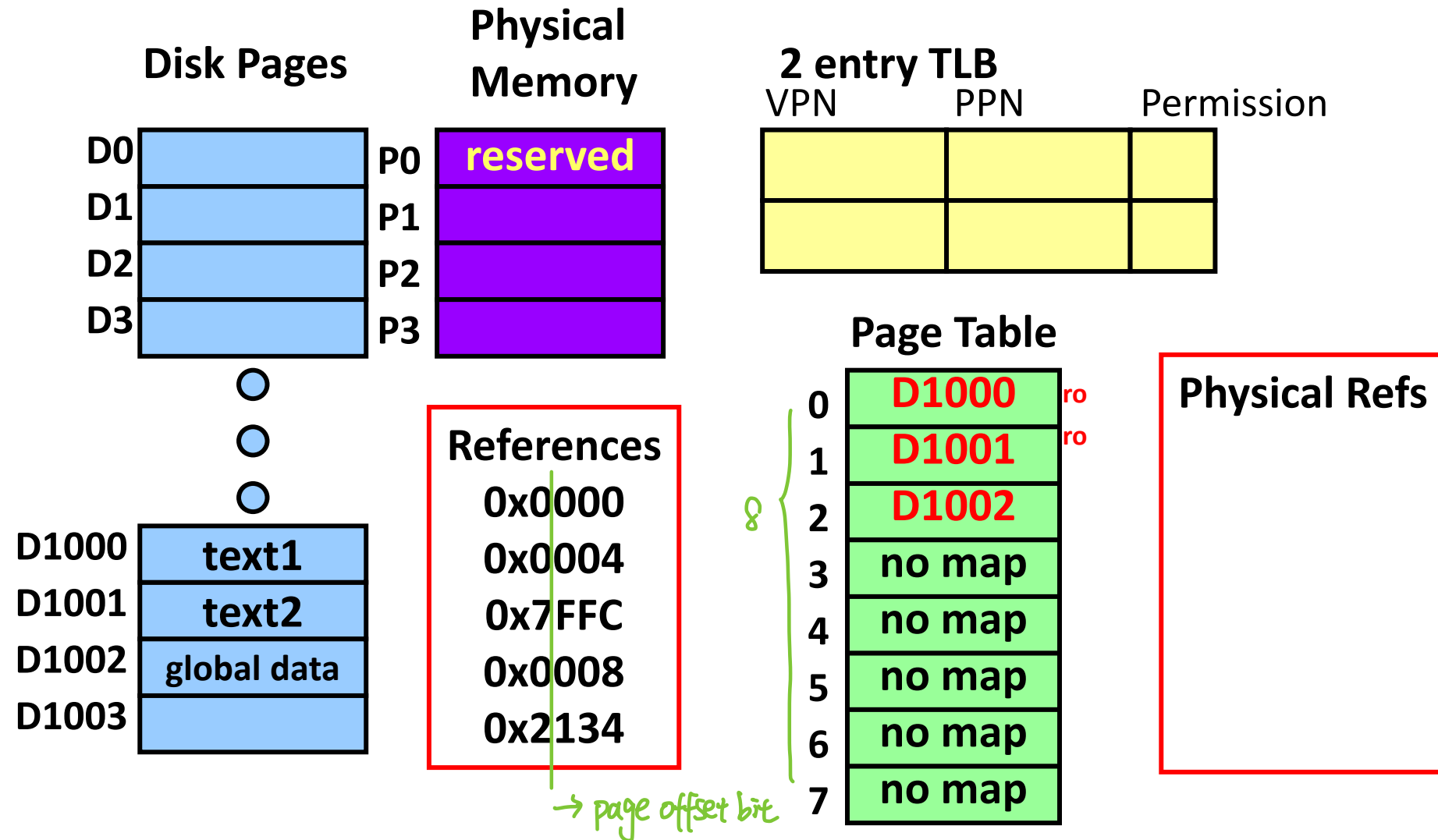
- Table Look-aside Buffers (TLB)
- **Virtual Memory Walkthrough**
- Cache Placement

Loading a program into memory

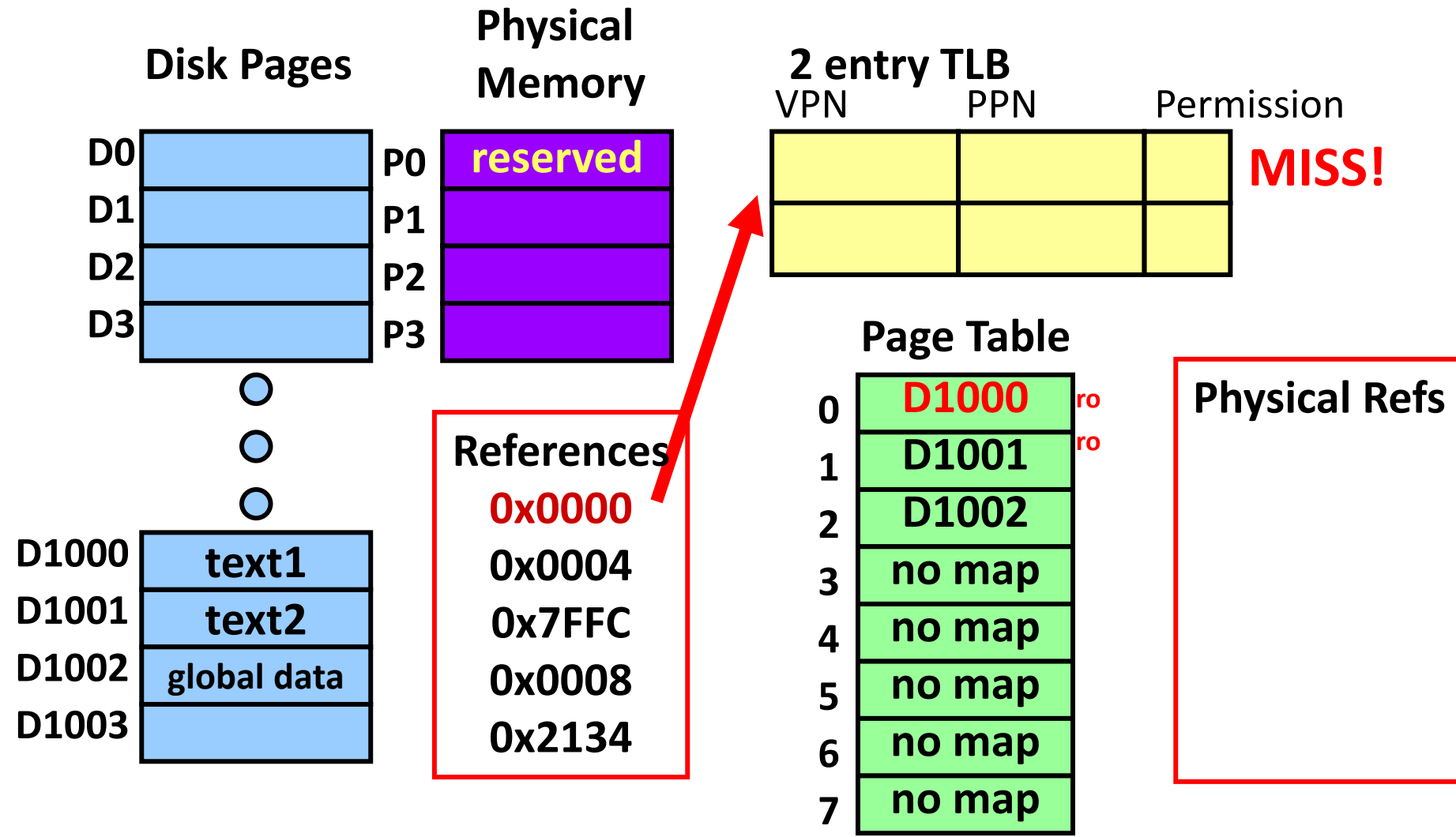
- Page size = 4 KB, Page table entry size = 4 B
- Page table register points to physical address 0x0000



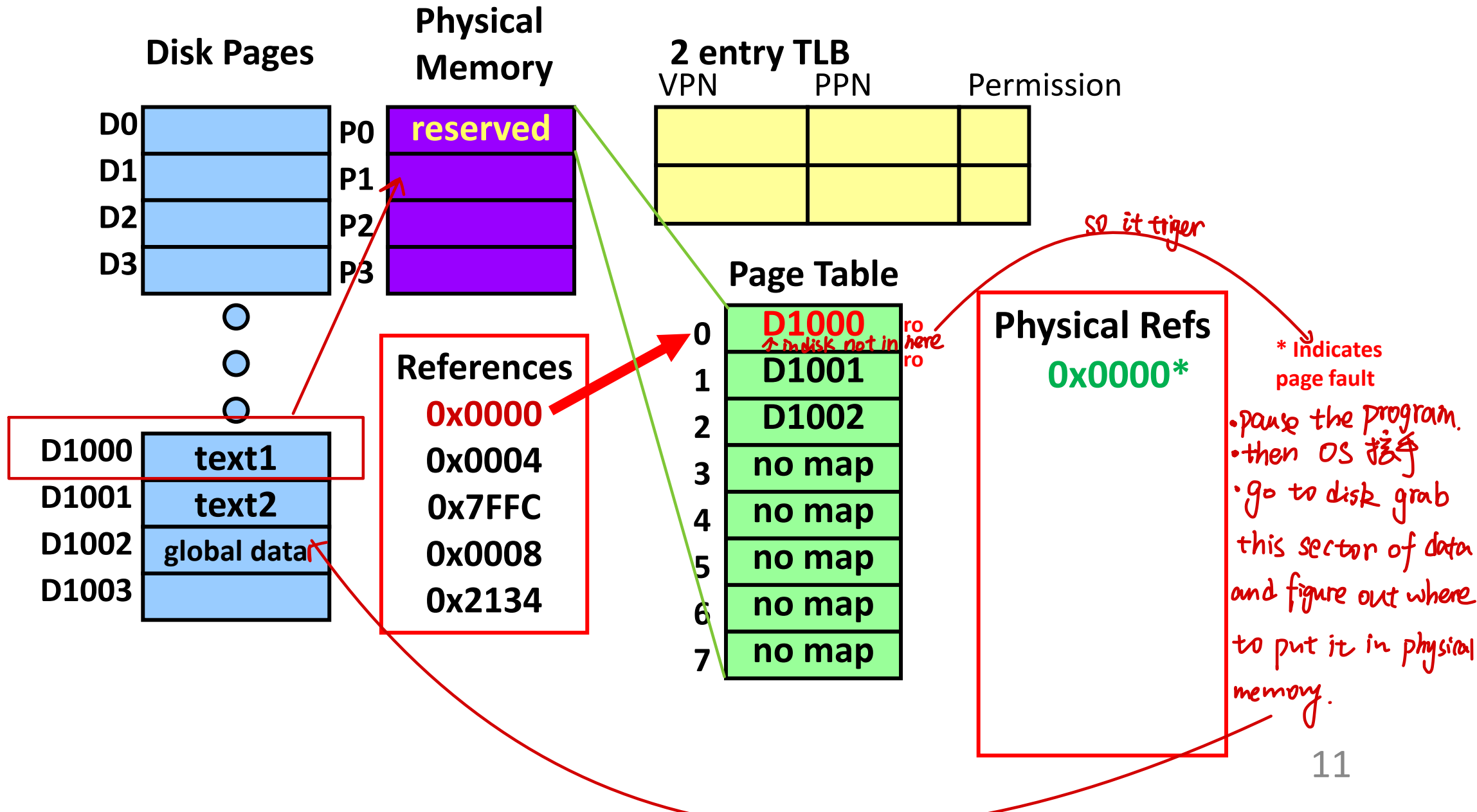
Step 1: Read executable header & initialize page table



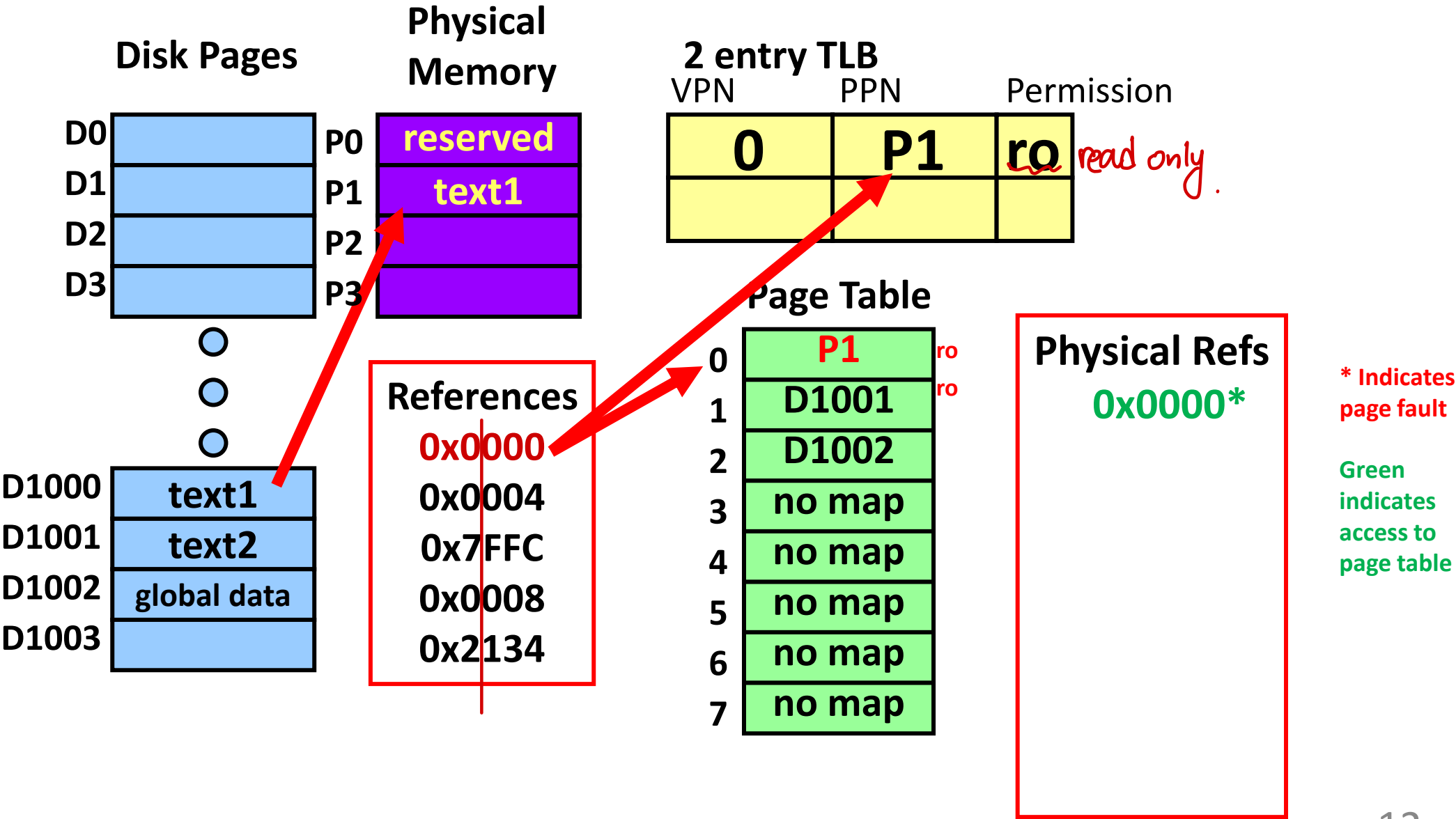
Step 2: Load PC from header & start execution



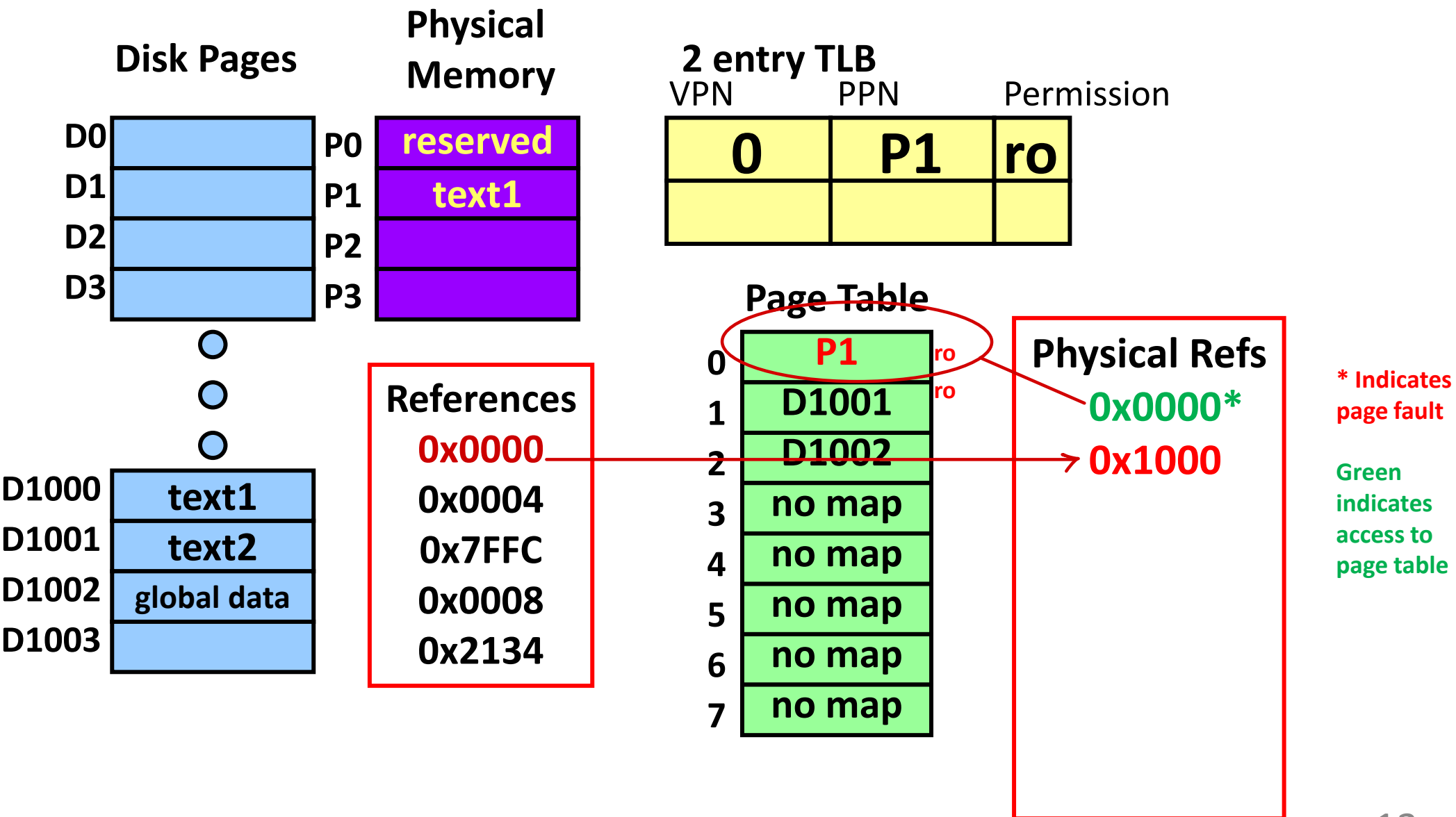
Fetching instruction 0000



Fetching instruction 0000



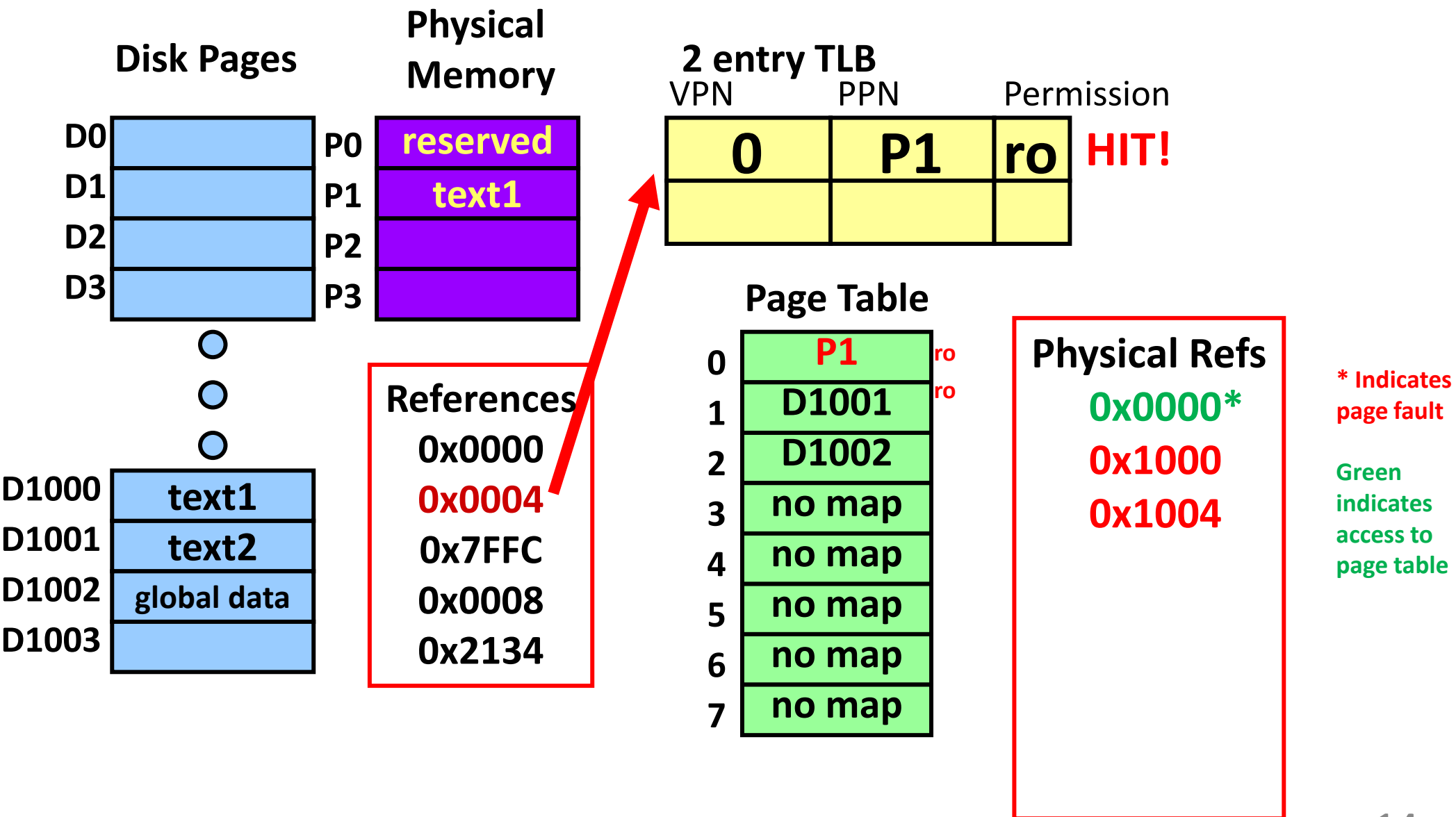
Fetching instruction 0000



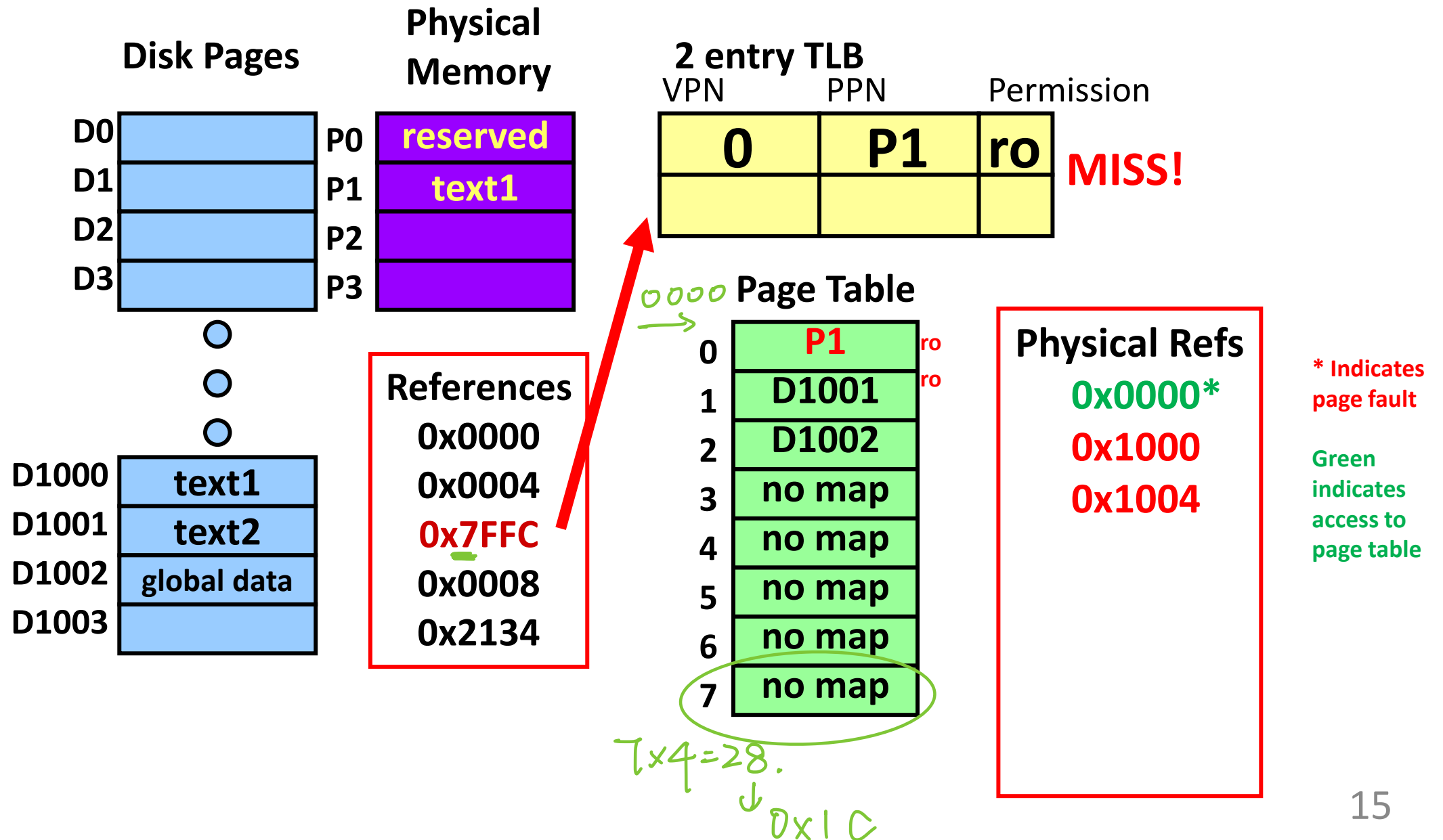
* Indicates page fault

Green indicates access to page table

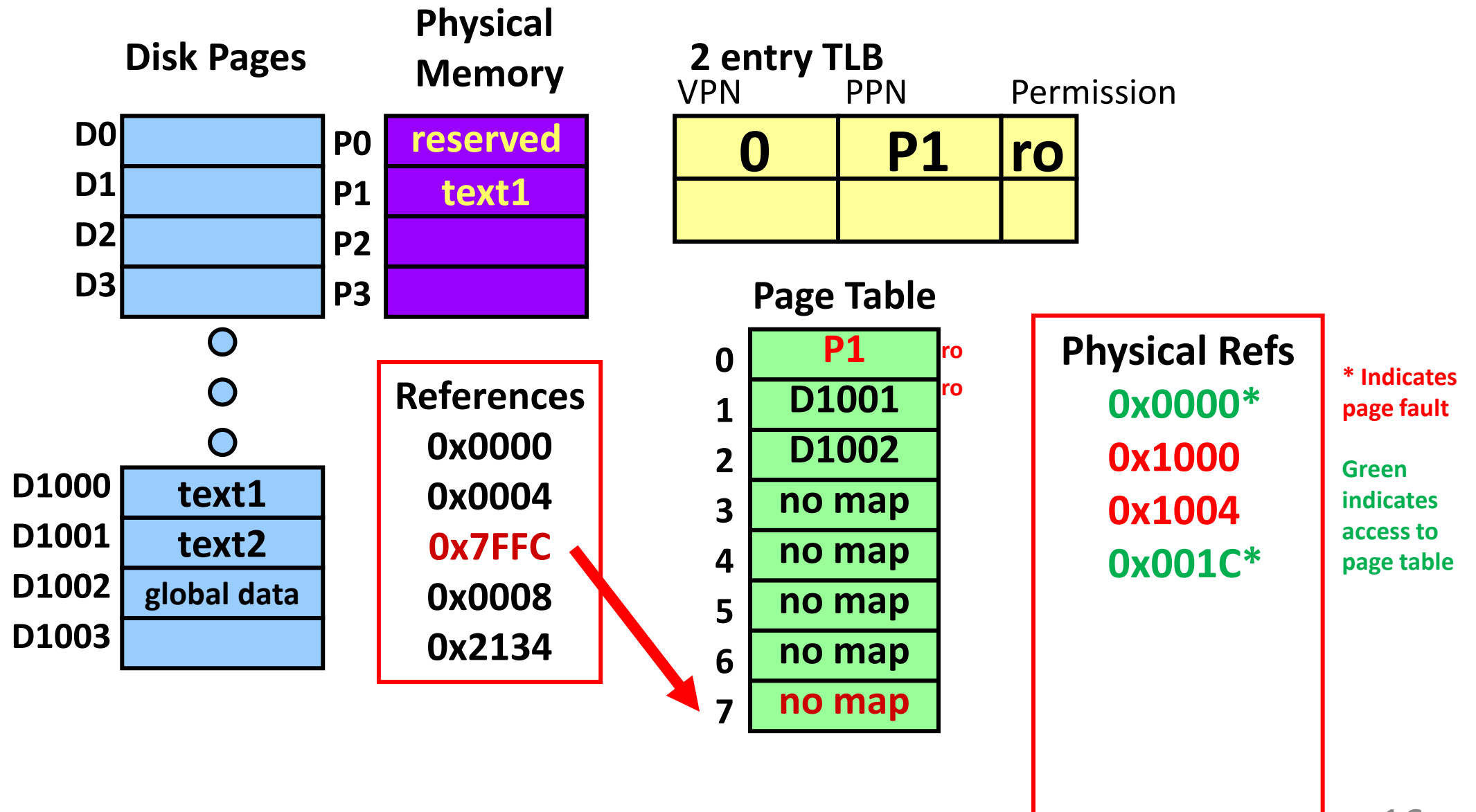
Fetching instruction 0004



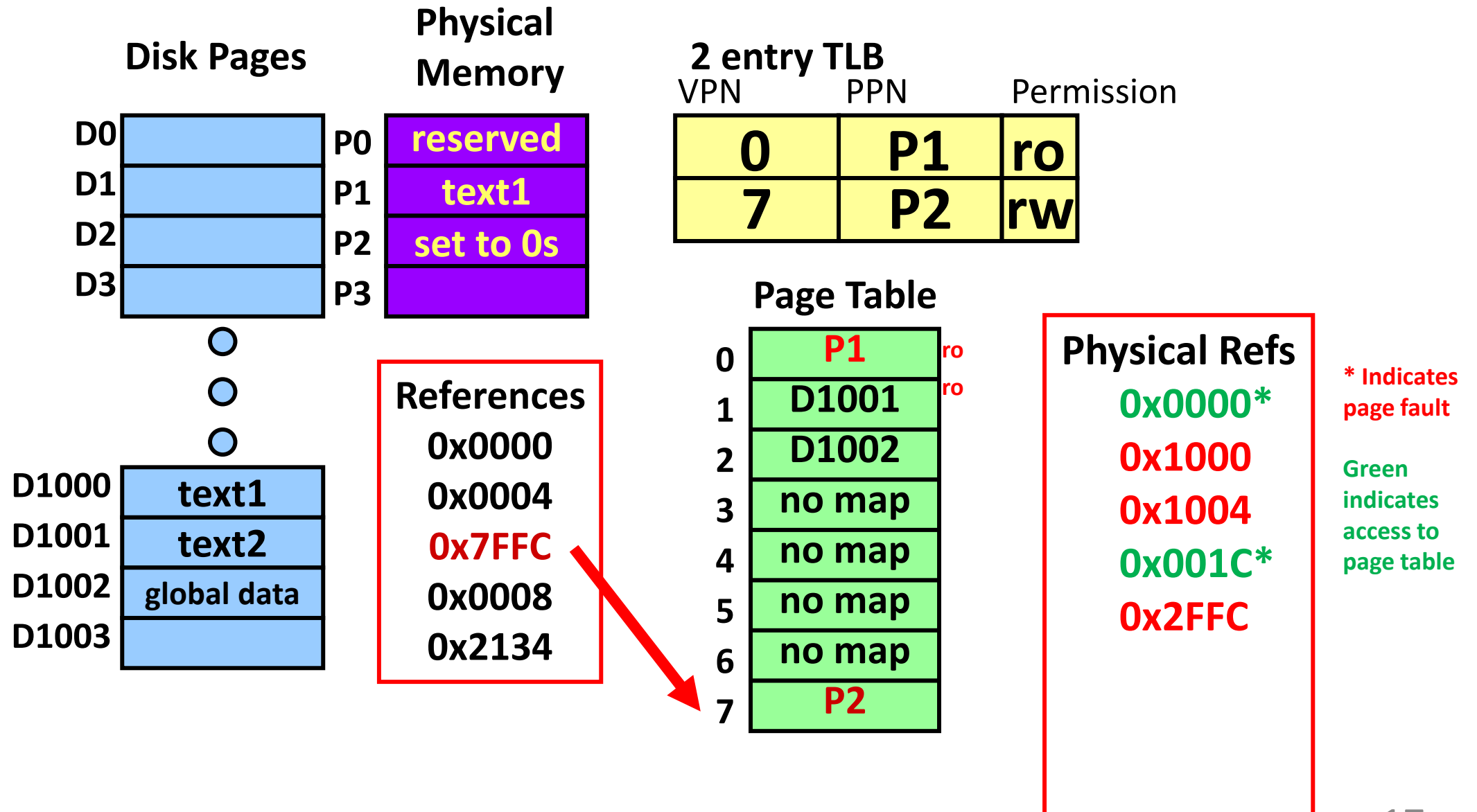
Reference 7FFC



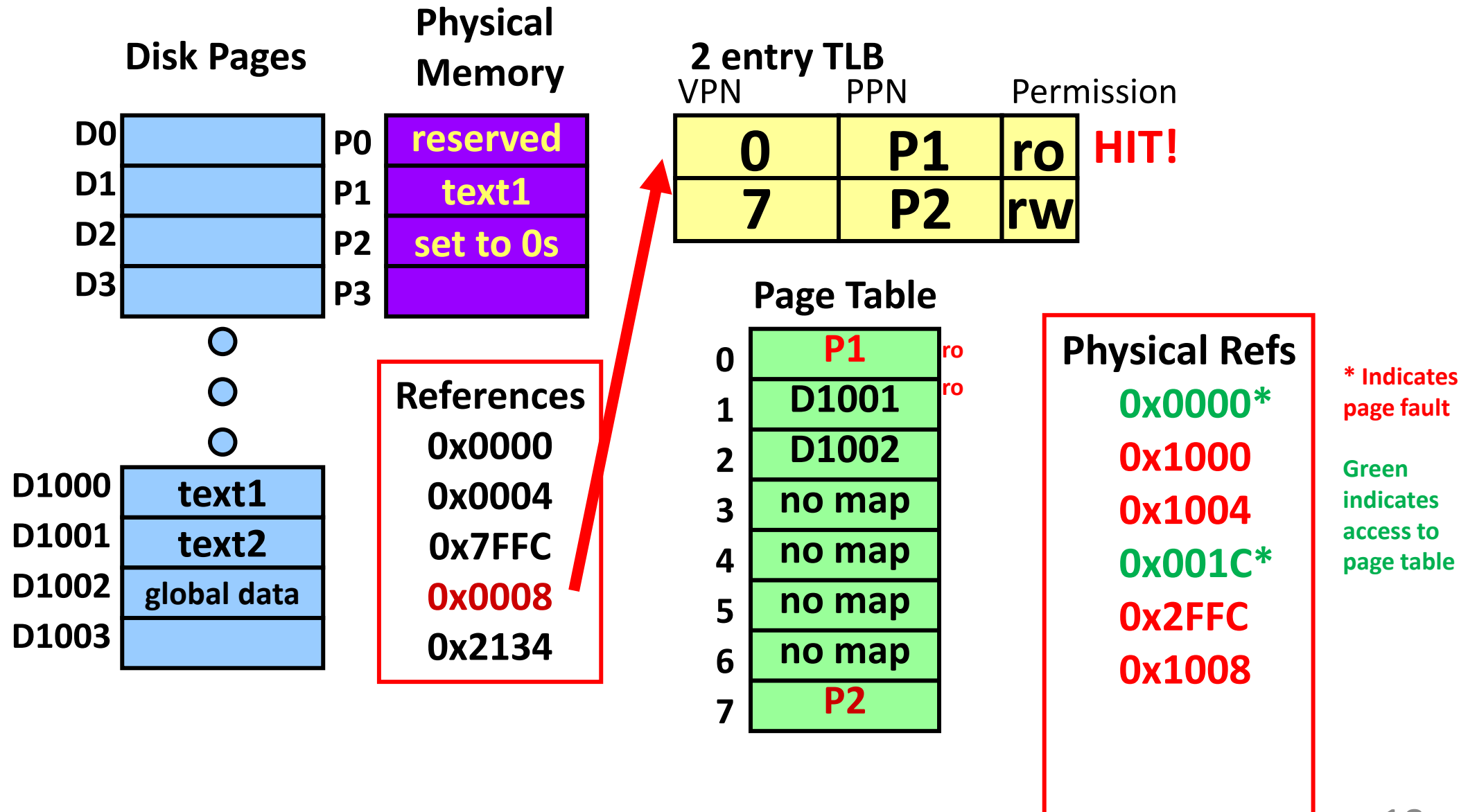
Reference 7FFC



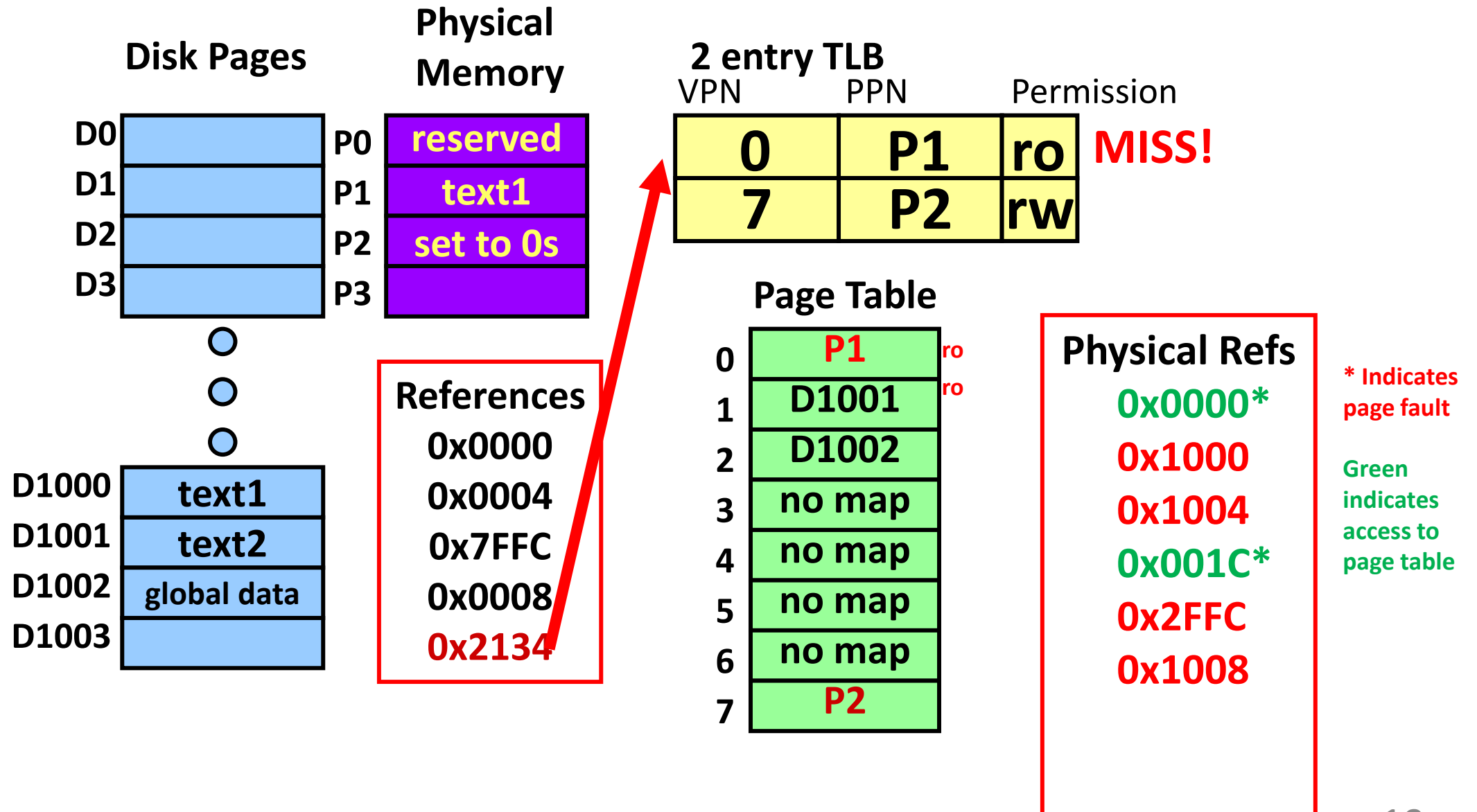
Reference 7FFC



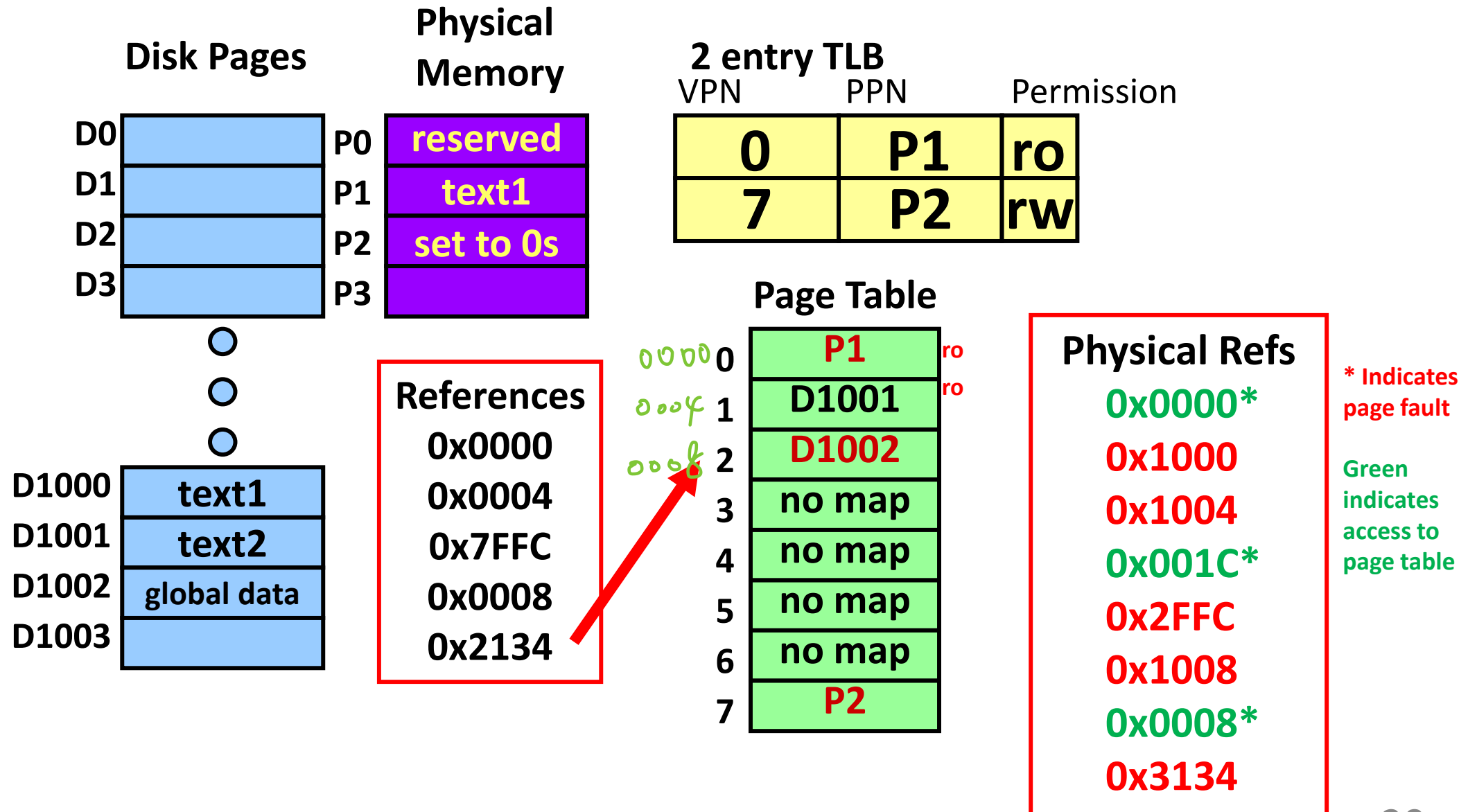
Fetching instruction 0008



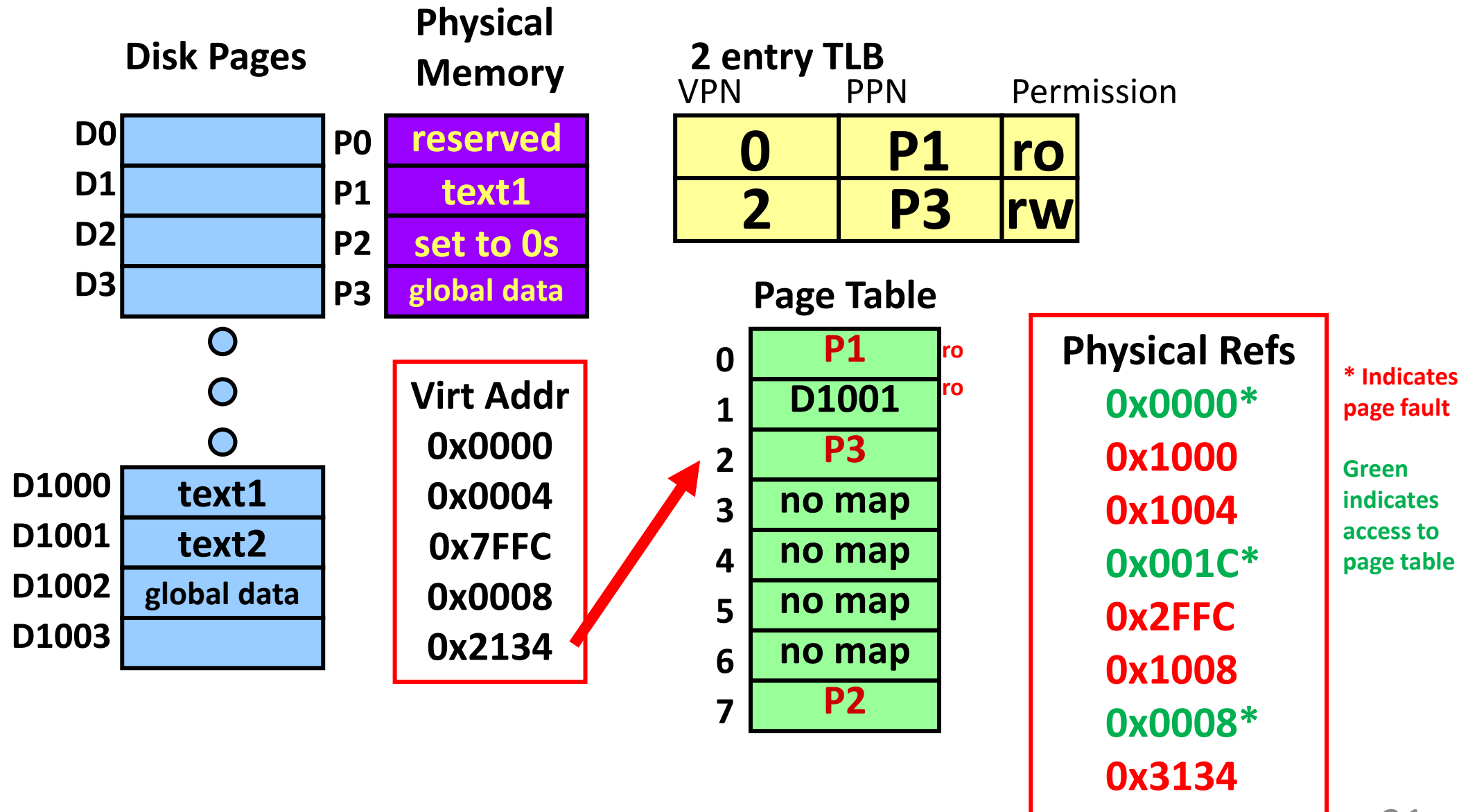
Reference 2134



Reference 2134



Reference 2134



Agenda

- Table Look-aside Buffers (TLB)
- Virtual Memory Walkthrough
- **Cache Placement**

Next topic: Placing Caches in a VM System

- VM systems give us two different addresses:
 - virtual and physical
- Which address should we use to access the data cache?
 - Physical address (after VM translations).
 - We have to wait for the translation; slower.
 - Virtual address (before VM translation).
 - Faster access.
 - More complex. *← multiple programs work together
And they share the cache*

Poll: Which would be faster to access?

- a) Address cache with virtual address
- b) Address cache with physical address

Cache & VM Organization: Option 1

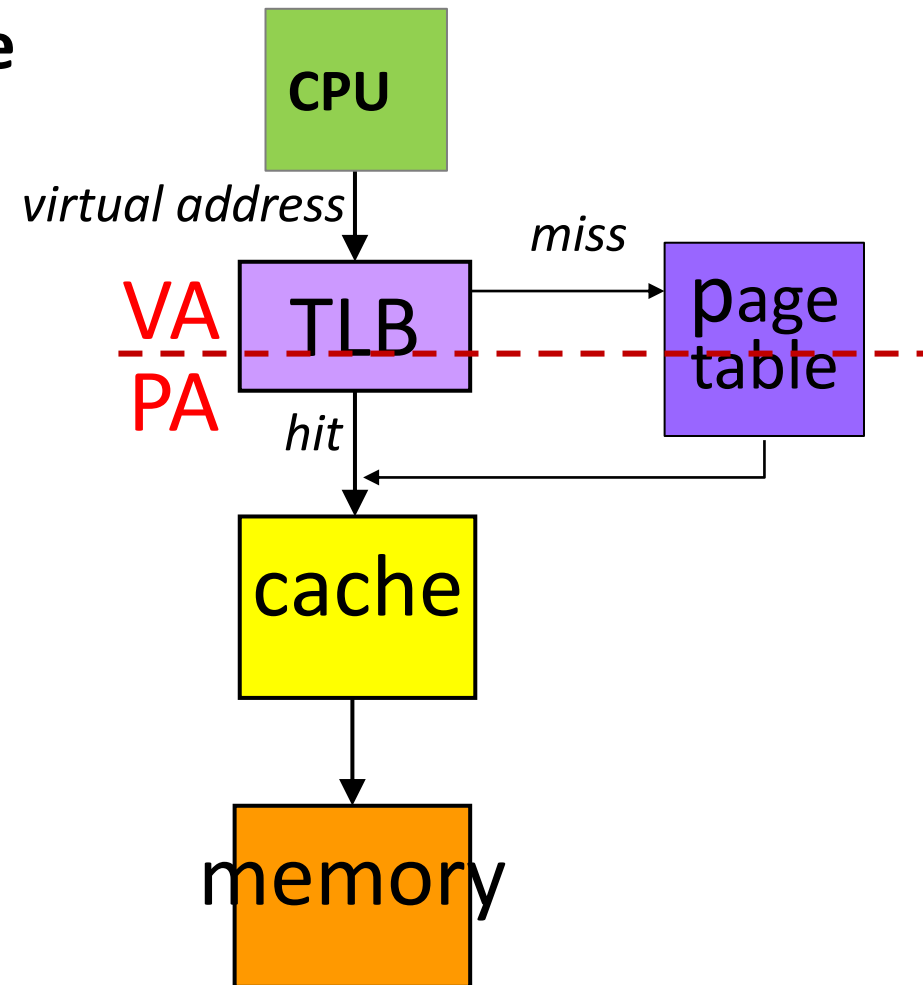
Physically-addressed Cache



Slower



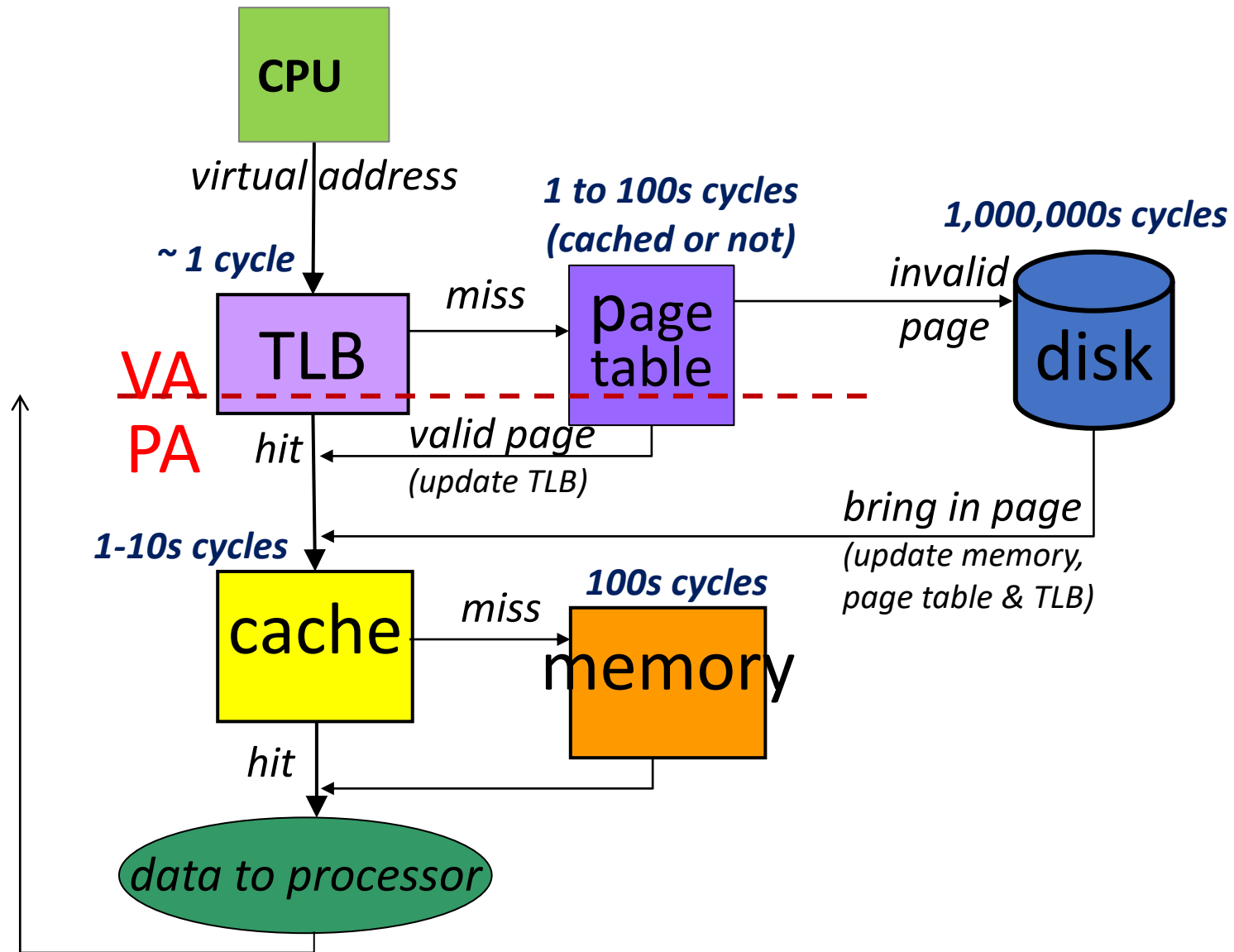
Low complexity



Physically addressed caches

- Perform TLB lookup **before** cache tag comparison
 - Use bits from **physical** address to access cache (tag, set index, and block offset bits)
- Slower access?
 - Tag lookup takes place **after** the TLB lookup
- Simplifies some VM management
 - When switching processes, TLB must be invalidated, but cache OK to stay as is
 - Implications? Might result in fewer cache misses if context switches very common (but they generally are not)

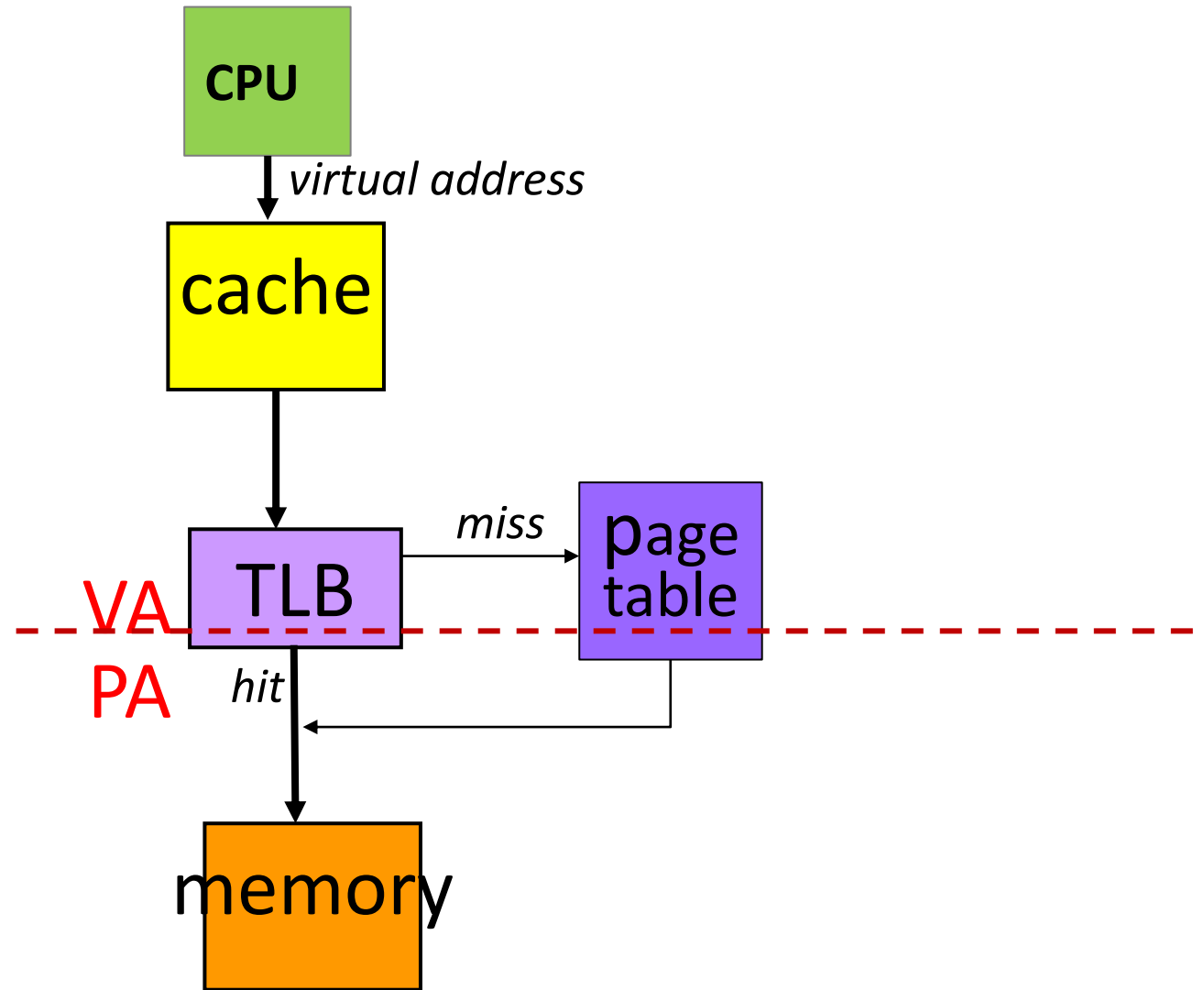
Physically addressed caches: detailed flow



Cache & VM Organization: option 2

Virtually-addressed Cache

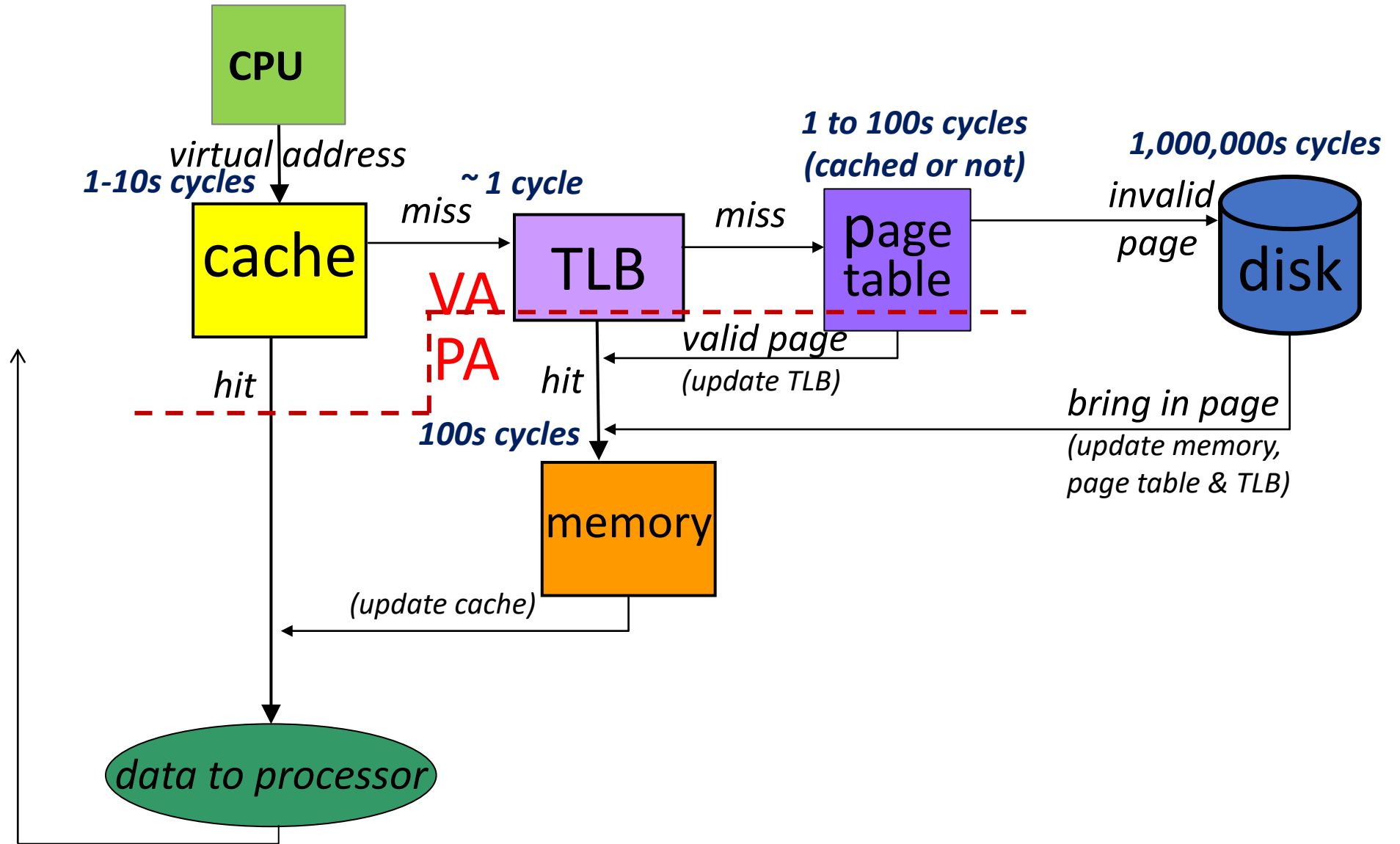
- ✗ High complexity (*aliasing*)
- ✓ Faster



Virtually addressed caches

- Cache uses bits from the virtual address to access cache (tag, set index, and block offset)
- Perform the TLB only if the cache gets a miss.
- Problems:
 - Aliasing: Two processes may refer to the same physical location with different virtual addresses (synonyms)
我们希望他们得到相同的结果, 但因为其 virtual address 不同, 则在 cache 中得到的结果也不同.
 - Two processes may have same virtual addresses with different physical addresses (homonyms)
 - When switching processes, TLB must be invalidated, dirty cache blocks must be written back to memory, and cache must be invalidated to solve homonym problem

Virtually addressed caches: detailed flow



OS Support for Virtual Memory

- It must be able to modify the page table register, update page table values, etc.
- To enable the OS to do this, **BUT** not the user program, we have different execution modes for a process.
 - **Executive** (or **supervisor** or **kernel** level) permissions and
 - **User level** permissions.

End of Virtual Memory

Extra Problems



Exercise using previous multi-level VM

Virtual Address	1 st level	2 nd level	Page offset	Page fault?	Physical page num.	Physical Address
0x000F0C						
0x001F0C						
0x020F0C						

1st level = 7b	2nd level = 8b	Page offset = 9b
----------------------------------	----------------------------------	-------------------------

Virtual address = 24b

Physical page number = 9b	Page offset = 9b
----------------------------------	-------------------------

Physical address = 18b

Assume memory for page tables are “somewhere else” in memory

Exercise using previous multi-level VM

Virtual Address	1 st level	2 nd level	Page offset	Page fault?	Physical page num.	Physical Address
0x000F0C	0x00	0x07	0x10C	Y	0x000	0x0010C
0x001F0C						
0x020F0C						

1 st level = 7b	2 nd level = 8b	Page offset = 9b
----------------------------	----------------------------	------------------

Virtual address = 24b

Physical page number = 9b	Page offset = 9b
---------------------------	------------------

Physical address = 18b

Assume memory for page tables are “somewhere else” in memory

Exercise using previous multi-level VM

Virtual Address	1 st level	2 nd level	Page offset	Page fault?	Physical page num.	Physical Address
0x000F0C	0x00	0x07	0x10C	Y	0x000	0x0010C
0x001F0C	0x00	0x0F	0x10C	Y	0x001	0x0030C
0x020F0C						

1 st level = 7b	2 nd level = 8b	Page offset = 9b
----------------------------	----------------------------	------------------

Virtual address = 24b

Physical page number = 9b	Page offset = 9b
---------------------------	------------------

Physical address = 18b

Assume memory for page tables are “somewhere else” in memory

Exercise using previous multi-level VM

Virtual Address	1 st level	2 nd level	Page offset	Page fault?	Physical page num.	Physical Address
0x000F0C	0x00	0x07	0x10C	Y	0x000	0x0010C
0x001F0C	0x00	0x0F	0x10C	Y	0x001	0x0030C
0x020F0C	0x01	0x07	0x10C	Y	0x002	0x0050C

1 st level = 7b	2 nd level = 8b	Page offset = 9b
----------------------------	----------------------------	------------------

Virtual address = 24b

Physical page number = 9b	Page offset = 9b
---------------------------	------------------

Physical address = 18b

Assume memory for page tables are “somewhere else” in memory

What's next for You?!

- What classes should you look at if your interested in this kind of stuff?

More Classes?

- EECS 470 – Computer Architecture
- Picks up where we leave off in 370
- Discuss more sophisticated enhancements to processor design
 - Out-of-order execution
 - Multi-core / multi-threaded processors
- Major Design Experience / Capstone
 - Work in teams to design an actual processor (not simulator) over the course of the semester
 - **LOT'S** of work
- Requires EECS 270
 - For Verilog (a Hardware Description Language)

More Classes?

- EECS 373 – Embedded Systems
 - Learn about parts of computer systems that aren't in the processor or cache
 - Input/output
 - Timers
 - Bus interfaces
 - Followed up by 473 (MDE / Capstone)
- EECS 471 – Applied Parallel Programming with GPUs
 - How are graphics processing units different than normal processors?
 - How do we take advantage of their raw power when writing software?

More Classes?

- EECS 427 – VLSI Design
 - Design a processor at the circuit level
 - Actually layout all the transistors
 - Less architecture – more circuit design
- EECS 570 (Parallel Computer Architecture) + 573 (Microarchitecture)
 - Learn about more current research into architecture
 - Requires 470

More Classes?

- EECS 483 (Front-end compilers) + 583 (Back-end compilers)
 - 583 is more relevant to architecture
 - How to design compilers to write efficient assembly?
 - Requires knowledge of hardware optimizations
- EECS 482 – Operating Systems
 - How do moderns systems support multiple threads running simultaneously?
 - How does OS manage virtual memory?
 - How do file systems work?
- EECS 388 – Computer Security
 - How do things like caches "leak" information about a program's data?
 - How can call stacks be tricked into executing arbitrary code?

More Classes?

- EECS 498-001 – Quantum Computing for the Computer Scientist
 - Winter 23
 - How can we redesign the computing stack to take advantage of the bizarre rules of quantum mechanics?
 - Rather than having bits be 0 or 1, have quantum bits be in **superpositions** of 0 and 1
 - Perform massive number of computations simultaneously... sort of
 - Requires 370

Other questions?!